DOCUMENTAȚIE

TEMA *1 – calculator de polinoame*

NUME STUDENT: MIHNEA ADRIANA - MARIA

GRUPA: 30226

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc95297886)

[3. Proiectare 3](#_Toc95297887)

[4. Implementare 3](#_Toc95297888)

[5. Rezultate 3](#_Toc95297889)

[6. Concluzii 3](#_Toc95297890)

[7. Bibliografie 3](#_Toc95297891)

# Obiectivul temei

Calculatorul de Polinoame este o aplicație care permite utilizatorilor să efectueze operații aritmetice cu polinoame. Acesta oferă opțiuni pentru adunare, scădere, înmulțire, împărțire, derivare și integrare. Scopul aplicației este de a face calculele cu polinoame mai eficiente și mai precise.

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Aplicația este limitată la efectuarea operațiilor cu unul sau două polinoame, care sunt introduse de către utilizator în casetele de text polinom1 și polinom2, sub forma

ax^n+bx^(n-1)+...+cx^2+dx^1+e.

Calculatorul de polinoame poate efectua :

- Adunarea (polinom1 +polinom2)

- Scăderea (polinom1 – polinom 2)

- Înmulțirea (polinom1 \* polinom 2)

- Împărțirea (câtul și restul împărțirii polinom1 / polinom 2)

- Derivata (polinom1’)

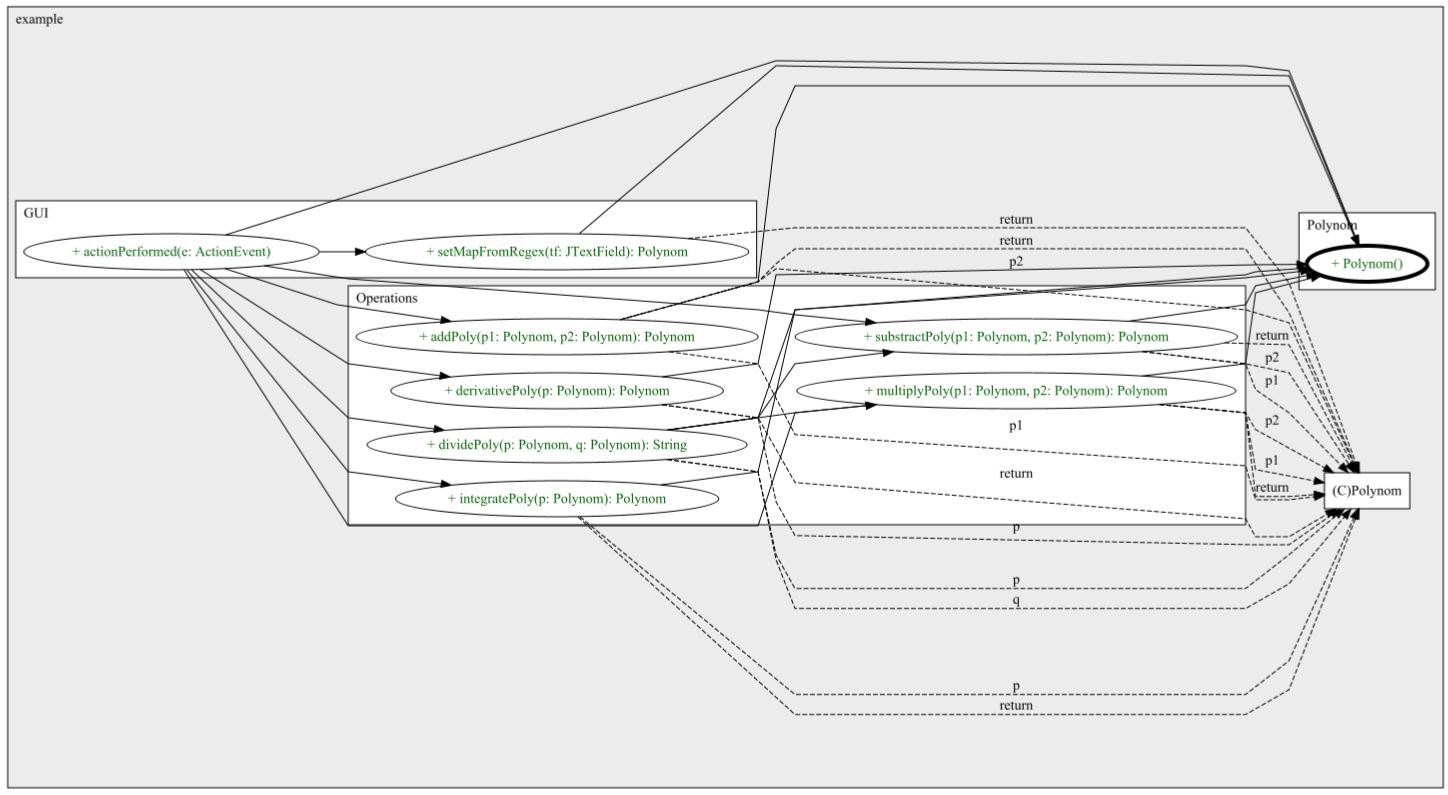
- Integrala (∫polinom1)

În cazul operațiilor de derivată și integral, input-ul va fi luat din caseta text polinom1, iar caseta polinom2 va fi ignorată.

# Proiectare

Ca și structură de date a fost folosit hashmap-ul polynomList<Integer, Double> cu ajutorul căreia i se asociază un anumit coeficient fiecărei puteri. De exemplu, 3x^2 va fi memorat in hashmap ca având valoarea 3 (coeficientul) la cheia 2 (puterea).

Diagrama UML :



# Implementare

###### Polynom :

1. Variabile
   1. Map<Integer, Double> polynomList – fiecărui monom al polinomului îi corespunde un câmp, cheia reprezentând puterea iar valoarea coeficientul
2. Metode
   1. Constructor – creează o nouă colecție de tipul polynomList
   2. printPoly – metodă folosită la depanare pentru a vedea cum arată rezultatul înainte de a-l trimite în interfața grafică
   3. toString – se iterează lista de monoame, atat puterea cât și coeficientul sunt convertite în string, este ștearsă partea de „.0” în cazul unui coeficient a cărui valoare este întreagă, iar apoi puterea și coeficientul se adaugă la string-ul result, tratând cazurile în care puterea este 0 sau 1

###### Operations :

Metode :

1. addPoly

* parametri : polinoamele p1 și p2
* este creat un nou polinom result în care va fi memorat rezultatul
* se iterează prin toate cheile posibile din hashmap, se verifică dacă există, caz în care se adună valorile (coeficienții) corespunzătoare acelor chei (puteri)
* rezultatul se pune in polynomList la cheia corespunzătoare, iar la final este returnat polinomul final

1. substractPoly

* funcționează exact la fel ca adunarea, dar în acest caz se scad coeficienții corespunzători puterilor

1. multiplyPoly

* parametri : polinoamele p1 și p2
* este creat un nou polinom result în care va fi memorat rezultatul
* se iterează imbricat prin cele două polinoame astfel încât să se înmulțească fiecare valoare a unei chei cu toate celelalte existente
* este tratat cazul în care la o anumită cheie încă nu este memorată nicio valoare, fiind prinsă excepția de tipul null pointer
* este returnat polinomul care conține rezultatul

1. dividePoly

* parametri: polinoamele p și q
* sunt create noi polinoame în care să fie memorat câtul și restul
* hashmap-ul polynomList este convertit într-un treemap, din cauza faptului că este necesar să fie luate valorile de la cheia (puterea) cea mai mare spre cea mai mică a polinoamelor
* atâta timp cât gradul polinomului p este mai mare decât gradul polinomului q, este pregătită noua putere și noul coeficient, sunt puse în cât și într-un obiect de tip polynom temporar ce servește înmulțirii cu polinomul q, iar rezultatul înmulțirii se scade din polinomul p
* în urma acestei bucle, quotient și remainder vor memora câtul și restul împărțirii, care vor fi returnate sub formă de string astfel încât să poată fi afișate în interfața grafică

1. derivativePoly

* parametru : polinomul p
* este creat un nou polinom result în care va fi memorat rezultatul
* se iterează prin polynomList a lui p și se calculează derivata, modificând întotdeauna valoarea de la cheia anterioară
* este tratat cazul in care puterea este 0, cand valoarea respective va fi ștearsă cu totul din hashmap

1. integratePoly

* parametru : polinomul p
* este creat un nou polinom result în care va fi memorat rezultatul
* se iterează polynomList a lui p, în variabila coef este calculat coeficientul la o anumită cheie în urma aplicării integralei, este rotunjit la o zecimală în cazul în care nu este număr întreg, iar apoi este pus la cheia potrivită în hashmap-ul rezultatului

###### GUI :

Metode :

* 1. Constructor – creează interfața grafică formata din mai multe elemente:
* JFrame – fereastra care se deschide la rularea programului, al cărei titlu este „Calculator polinomial”
* 3 obiecte de tip JPanel pe care se aranjează corespunzător etichetele, casetele de text și butoanele
* 3 obiecte de tip JLabel – etichetele ce arată cărei casete îi corespund cele 2 polinoame de input si polinomul de rezultat
* 3 obiecte de tip JTextField corespunzătoare celor 3 etichete, unde utilizatorul poate introduce date
* 6 butoane, corespunzătoare celor 6 operații pe care le poate efectua calculatorul
* Butoanele sunt aranjate în formatul GridLayout, astfel că, indiferent în ce mod se redimensionează fereastra, ele vor păstra un aspect estetic
* Cele 2 casete de text corespunzătoare polinoamelor de input au fost deja completate cu niște exemple, însă ele pot fi modificate în orice moment de către utilizator

b. setMapFromRegex – metodă ce pune conținutul casetelor text în hashmap-uri

1. parametru – caseta de text din care se ia string-ul
2. textul este memorat în stringPoly
3. polyPattern este un regex specific ce împarte stringPoly în monoame
4. având în vedere că fiecare monom are o formă precum ax^n, a fost folosit vectorul de string-uri coefAndPower care memorează la poziția 1 puterea, iar la poziția 0 coeficientul, separate tot prin intermediul unui regex
5. sunt tratate cazurile speciale precum puterea 0 și sunt interceptate excepții care provocau terminarea programului (ex. lipsa coeficientului)
6. puterea și coeficientul sunt puse la locul potrivit în hashmap-ul polinomului poly care este în cele din urmă returnat

c. actionPerformed – metodă implementată de interfața ActionListener

* face legătura între butoane și metodele din clasa Operations
* atunci când este apăsat un anumit buton, este apelată operația corespunzătoare iar rezultatul ei este pus în caseta text tf3

###### Main :

Apelează constructorul de GUI și astfel declanșează pornirea aplicației

# Rezultate

Au fost implementate metode de test pentru fiecare operație prezentă în clasa Operations. Clasa OperationsTest prelucrează polinoamele p1 și p2 pentru fiecare test.

Înainte de a fi pornite testele, se apelează metoda initialisePolynomials cu label-ul @BeforeClass, care apeleaza constructorul clasei Polynom si populează hashmap-urile cu niște valori arbitrare pentru testare.

În interiorul fiecărei metode de test este construit un polinom res care este populat cu rezultatul așteptat, iar apoi se compară rezultatul fiecărei operații cu rezultatul așteptat corespunzător.

# Concluzii

În urma efectuării acestei teme au fost fixate cunoștințele dobândite la materia Programare Orientată pe Obiect, precum lucrul cu colecții, regex-uri, design-ul specific OOP.

Ca și posibile dezvoltări ulterioare pot fi amintite îmbunătățirea interfeței grafice, extinderea datelor de intrare la mai mult de 2 polinoame sau un rezultat mai precis.

# Bibliografie

1. Folosirea expresiilor regulate - https://stackoverflow.com/questions/36490757/regex-for-polynomial-expression
2. Verificarea regex-urilor - <https://regex101.com/>
3. Junit - <https://www.digitalocean.com/community/tutorials/junit-setup-maven>
4. Java Swing - https://www.javatpoint.com/java-swing